**Abstract**

This project aims to create a robust and user-friendly inventory management system for a retail store. The primary goal is to provide a comprehensive tool that allows a store manager to accurately track stock levels, monitor sales trends, and automate the reordering process. The system will be built to reduce manual effort, minimize human error, and provide real-time data to improve decision-making, ultimately leading to better customer satisfaction and increased profitability. The core functionalities will include adding and updating products, recording sales, generating inventory reports, and alerting the manager to low-stock items.

**Development**

**1. Technology Stack Selection**

* **Frontend:** We will use a modern JavaScript framework like **React** or **Vue.js** for building a dynamic and responsive user interface. This will ensure a smooth user experience on various devices, from desktops to tablets.
* **Backend:** A backend framework like **Node.js with Express.js** or **Python with Flask** will be used to handle business logic, manage API endpoints, and interact with the database. These choices offer a good balance of performance and development speed.
* **Database:** A relational database such as **PostgreSQL** or a NoSQL database like **MongoDB** will be used to store product, sales, and supplier information. A relational database is generally preferred for inventory systems due to the structured nature of the data and the need for data integrity.

**2. System Architecture**

The system will follow a **three-tier architecture**:

* **Presentation Tier (Frontend):** The user interface where the store manager and staff interact with the system. This includes dashboards, forms for adding products, and screens for processing sales.
* **Application Tier (Backend):** The server-side logic that processes requests from the frontend, performs calculations, and interacts with the database. This layer will handle tasks like inventory updates, sales calculations, and report generation.
* **Data Tier (Database):** The database where all the application's data is stored. It will include tables or collections for products (with fields like SKU, name, quantity, price), sales transactions, and suppliers.

**3. Key Features to Develop**

* **Product Management:** Functionality to add new products, edit existing product details (e.g., price, description), and remove products.
* **Sales Tracking:** A module to record sales transactions. When a sale is made, the system will automatically decrement the quantity of the sold item from the inventory.
* **Inventory Reports:** The system will generate various reports, such as:
* **Current Stock Report:** A real-time list of all products and their current quantities.
* **Low Stock Report:** An automatic alert for products that have fallen below a predefined minimum threshold.
* **Sales Report:** A summary of sales over a specific period (daily, weekly, monthly).
* **Search and Filter:** The ability to quickly search for products by name or SKU and filter the inventory list based on criteria like category or stock level.
* **User Authentication:** A secure login system to ensure only authorized personnel can access the inventory data.

**Coding**

**1. Database Schema (PostgreSQL example)**

A simple schema might include a products table:

SQL

CREATE TABLE products (  
    product\_id SERIAL PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    sku VARCHAR(50) UNIQUE NOT NULL,  
    quantity INTEGER NOT NULL CHECK (quantity >= 0),  
    price DECIMAL(10, 2) NOT NULL,  
    category VARCHAR(100),  
    last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

This table stores the essential information for each product. The quantity field is critical and must be carefully managed with each sale and restock.

**2. Backend Logic (Node.js/Express.js example)**

An API endpoint to record a sale would involve a few steps:

* A POST request is sent to an endpoint like /api/sales with the product ID and quantity sold.
* The backend receives this request.
* It performs a database query to find the product by its ID.
* It then checks if the current quantity is greater than or equal to the quantity being sold.
* If enough stock is available, it updates the product's quantity in the database using an UPDATE statement, decrementing the value.
* It then records the transaction in a separate sales table for reporting purposes.
* A success or error message is sent back to the frontend.

**3. Frontend Interface (React example)**

The frontend will have components for different parts of the system. For the inventory list, a component might fetch data from the backend's /api/products endpoint and display it in a table.

JavaScript

import React, { useState, useEffect } from 'react';  
  
function InventoryList() {  
    const [products, setProducts] = useState([]);  
  
    useEffect(() => {  
        fetch('/api/products')  
            .then(res => res.json())  
            .then(data => setProducts(data));  
    }, []);  
  
    return (  
        <div>  
            <h2>Current Inventory</h2>  
            <table>  
                <thead>  
                    <tr>  
                        <th>Product Name</th>  
                        <th>SKU</th>  
                        <th>Quantity</th>  
                        <th>Price</th>  
                    </tr>  
                </thead>  
                <tbody>  
                    {products.map(product => (  
                        <tr key={product.product\_id}>  
                            <td>{product.name}</td>  
                            <td>{product.sku}</td>  
                            <td>{product.quantity}</td>  
                            <td>${product.price}</td>  
                        </tr>  
                    ))}  
                </tbody>  
            </table>  
        </div>  
    );  
}  
  
export default InventoryList;

This code snippet demonstrates how the frontend would display the current inventory by fetching data from the backend API. The project will continue to build upon these foundational coding principles to create the full-featured inventory system.